Syntactic/Semantic Analysis for High-Precision Math Linguistics

Jan Frederik Schaefer Michael Kohlhase

Friedrich-Alexander-Universität Erlangen-Nürnberg

Abstract

There have been some impressive advancements in the field of natural language processing, mostly with the use of statistical methods. We believe that statistical methods are inherently limited in their accuracy, restricting their usefulness especially in applications where high accuracy is indispensable. One such application is the translation and formalization of STEM documents, where small errors can render the results unusable. In this paper, we will present our ongoing work on the *Mathematical Grammatical Framework* (MGF), which is based on the *Grammatical Framework* (GF). The goal of MGF is to accurately translate and formalize mathematical texts. At this stage, its coverage is restricted to a small set of examples, which can be correctly translated between English and German, and formalized in a logical representation.

1 Introduction

There is a need for specialists in technical fields, who regularly have to go through the tedious process of finding, understanding and applying technical knowledge. We believe that the formalization of this knowledge would allows us to create a variety of tools that help increase the productivity of sought-after experts. This includes: definition look-up, semantic search, correctness/applicability checks, enhanced screen readers and accurate translations.

For now, we restrict our efforts to mathematical documents. In particular, the arXMLiv corpus [Arxa], which currently contains over 10^6 HTML5 documents generated from the Cornell e-print arXiv [Arxb] with LaTeXML [Mil]. Manual formalization would be an insurmountable challenge, so automated formalization appears to be the only solution.

As a motivating example, we chose the definition "*A positive integer n is called prime, iff there is no integer $1 < m < n$ such that $m|n$.*". It is sufficiently complex to exhibit some of the challenges we can face, while simple enough to serve as a first development milestone. Maybe the most obvious challenge is dealing with the formulae, which have different grammatical roles in the sentence. For example, "$1 < m < n$" corresponds to a noun phrase in an apposition, while "$m|n$" has the role of a complete subsentence (see also Section 3.2). For translations, German and English are supported. A translation to German of this example sentence could be "*Eine positive ganze Zahl n heißt Primzahl genau dann, wenn es keine ganze Zahl $1 < m < n$ gibt, sodass $m|n$.*". On a side note: The German word "*Primzahl*" is a single noun, while the corresponding English word "*prime number*" is composed of a noun ("*number*") and an adjective ("*prime*"). This poses some interesting challenges that we will have to look into in the future. Apart from translation, we are also interested in formalization. A logical representation of the prime number definition could be something like $\forall n.\mathbf{pos}(n) \wedge$

In: O. Hasan, F. Rabe (eds.): Proceedings of the CICM 2018, Hagenberg, Austria, 13-08-2018, published at http://ceur-ws.org

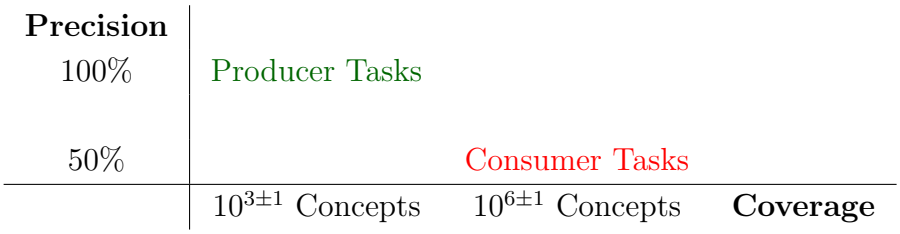


Figure 1: There are two types of tractable NLP tasks: High-coverage, low-precision *consumer tasks* and high-precision, low-coverage *producer tasks*.

$\mathbf{int}(n) \Rightarrow (\mathbf{prime}(n) \Leftrightarrow \neg\exists m.\mathbf{int}(m) \wedge \mathbf{less}(1,m) \wedge \mathbf{less}(m,n) \wedge \mathbf{divides}(m,n))$. Note that we interpreted the adjective "*positive*" intersectively – more on this in Section 3.3.1.

Aarne Ranta identifies two classes of tractable areas for natural language processing: *consumer tasks* and *producer tasks* [Ran16] (Figure 1). *Consumer tasks* require large coverage, usually achieved with statistical methods, but are inherently limited in their accuracy. A prominent example for this is full-range machine translation as for example with Google Translate. Mathematical documents, however, fall in the category of *producer tasks*: The coverage is restricted to a technical subset of a language with a domain-specific vocabulary. Such producer tasks can be tackled with Aarne Ranta's *Grammatical Framework* (GF) [Ran04; Ran11; GF].

In this paper, we are going to describe our ongoing efforts to create the *Mathematical Grammatical Framework* (MGF). It is based on GF and currently supports the translation and formalization of a small set of mathematical sentences. After providing an overview of related work and a brief introduction to GF in Section 2, we will describe our setup in more detail in Section 3. Section 4 provides an evaluation of the current state of our system, along with a few examples. Section 5 concludes the paper.

Acknowledgements

We are very grateful to Aarne Ranta, Magdalena Wolska and Deyan Ginev, whose expertise and insights greatly helped our efforts to create MGF.

2 Related Work and Preliminaries

Another project that uses GF for mathematical texts is the "Mathematical Grammar Library" [CS12; SX11; Arc+12]. It supports the translation of propositions like "*the absolute value of x is greater than x or equal to x*" in an impressive number of languages as well as the conversion to LaTeX. It is based on the WebALT project [Cap], which used GF to translate mathematical exercises for students into many languages. Both of these projects based their grammars on OpenMath's content dictionaries [Bus+04]. We believe that MGF can achieve wider discourse coverage without this restriction.

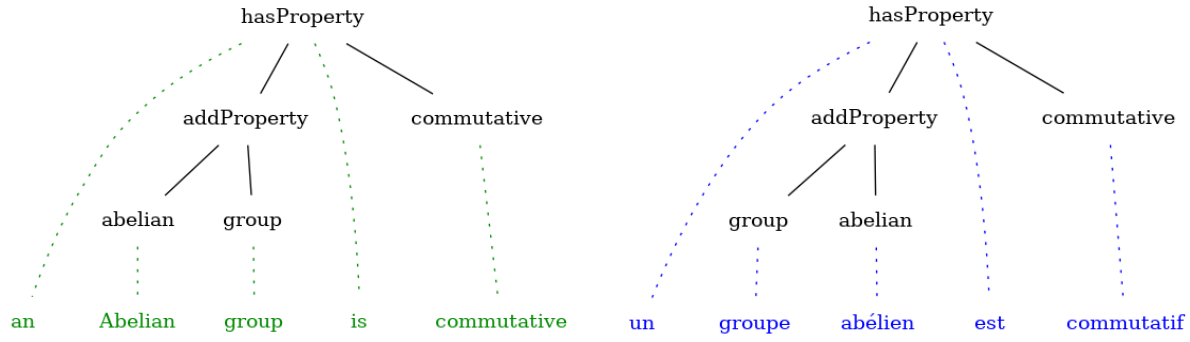
Other research has focused on informal proofs like Claus Zinn's work on the verification of informal textbook proofs [Zin04], in which he uses discourse representation theory [Kam81] for the formal representation of proofs, and Magdalena Wolska's work on informal proofs by students [Wol13; WKK04]. Some projects avoid challenges in informal language by introducing a controlled natural language, as for example the System for Automated Deduction (SAD) with its ForTheL language [VLP07].

An analysis of natural language in mathematics can be found in [Zin04], [Wol13] and [Bau99]. The structure (and parsing) of formulae has been investigated in [Gin11].

The work mentioned so far has mostly focused on the understanding and formalization of informal language. The other direction has been studied as well. A particularly interesting case is the generation of human-understandable representations of formal proofs [Hor99; Hua89].

2.1 Introduction to GF

GF (the Grammatical Framework) is "a programming language for multilingual grammar applications" [GF]. GF Grammars can be divided into two parts: The *abstract syntax* and the *concrete syntax(es)*. The *abstract syntax* describes what meanings can be expressed in the grammar. A small example abstract syntax module is shown in Listing 1. The *concrete syntax* (Listing 2) describes how these meanings are expressed in a particular language, i.e. it provides a mapping to and from strings. GF comes with a *resource grammar library* that provides basic



(a) Parsing English

(b) Linearizing into French

Figure 2: Translation from English into French.

syntactic rules for over 35 languages. This way, we can write grammars without dealing with language-specific syntax details as inflections, word order, etc.

Listing 1: Example abstract syntax module.

```
abstract Math = {
    cat Object; Property; Statement;
    fun
        group : Object;
        abelian : Property;
        commutative : Property;
        addProperty : Property -> Object -> Object;
        hasProperty : Object -> Property -> Statement;
}
```

Listing 2: Example concrete syntax module for English. We use the *resource grammar library* to avoid dealing with syntactic details.

```
concrete MathEng of Math = open SyntaxEng, ParadigmsEng in {
    lincat
        Object = CN; -- common noun
        Property = AP; -- adjective phrase
        Statement = S; -- sentence
    lin
        group = mkCN (mkN "group");
        abelian = mkAP (mkA "Abelian");
        commutative = mkAP (mkA "commutative");
        addProperty prop obj = mkCN prop obj;
        hasProperty obj prop = mkS (mkCl (mkNP aSg_Det obj) prop);
}
```

Now, we can take for example the English string "*an Abelian group is commutative*" and parse it with the concrete English syntax into its semantic tree representation (Figure 2a). If we also create a concrete syntax for the French language, we can use it to linearize this tree into a French string (Figure 2b). In other words, we translate a statement from English to French.

3 Using GF for Mathematical Documents

Mathematical Documents consist of natural language (discourse), formulae, diagrams, tables, images etc. We concentrate on the interplay of discourse and formulae, which so far has been little studied.

When creating a grammar, it is important to find the right balance between a semantic design and a syntactic one. In general, we aim at a semantic design which abstracts away from any particular language and only

focuses on the meaning. While this simplifies content formalization, there are some draw-backs, which require us to compromise and keep some syntactic elements. Consider the statement "f *is injective, but not surjective*". The meaning of it is something like $\textbf{injective}(f) \wedge \neg\textbf{surjective}(f)$, which could just as well be written as "f *is injective and not surjective*". During translation, it is desirable to maintain the contrast indicated by the word "*but*", which means that it has to be expressed in the parse trees.

Our grammar consists of two parts: a part for parsing formulae (Section 3.2) and a part for parsing discourse (Section 3.3). This division suggests itself, especially when it comes to the concrete syntaxes: For discourse, we need concrete syntaxes for different natural languages (English, German, ...), as well as for a logical representation. Formulae, on the other hand, are not very language dependent, but rather representation dependent (LaTeX, MathML, ...). There are, however, some formulae that have different representations in different languages. For example, the formula "$gcd(a, b)$" ("*gcd*" is the "*greatest common divisor*") translates to "$ggT(a, b)$" in German ("*ggT*" stands for "*größter gemeinsamer Teiler*").

The discourse vocabulary, as well as identifiers and operators in formulae are stored in lexica. So far, these have been mostly hand-written to cover a few examples. In the future, we can try to generate lexica of identifiers and operators from corpora. For the natural language vocabulary, multilingual resources (like SMGloM [Gin+16]) would be needed.

3.1 Representation of Mathematical Language

In natural language processing, the representation of sentences is usually rather straight-forward, as there are standardized ways to represent a sentence as a sequence of characters. In mathematics, however, this is more challenging, as formulae need to be represented as well.

Many mathematical papers are written in LaTeX. However, parsing LaTeX is not an easy task. As mentioned before, our work is focused on the arXMLiv corpus [Arxa], which consists of HTML5 documents generated from LaTeX with LaTeXML [Mil]. The formulae are represented as MathML [Aus+03] and come in two different flavours: *Content MathML* and *Presentation MathML*. Content MathML is a more semantic representation, which would theoretically make it the better choice for formalization. However, LaTeXML does not generate high-quality content MathML, as this is a highly non-trivial undertaking. This leaves us with presentation MathML. To avoid the hassle of parsing XML in GF, we decided to use a simplified representation based on the presentation markup. For example, the formula "$1 < m < n$" would be represented as `$ mrow( mn( 1 ) mo( < ) mi( m ) mo( < ) mi( n ) ) $`. It is easy to convert between this representation and presentation MathML.

Of course, it is always possible to create more concrete syntaxes to support other formula representations. For example, there is a different concrete syntax for logical representations. The example above e.g. would be represented as $\textbf{less}(1, m) \wedge \textbf{less}(m, n)$.

3.2 The Formula Grammar

As mentioned before, formulae can take different grammatical roles in a sentence, and based on that we parse them differently. Let's demonstrate that with the formula "$n > 2$". It could be a part of the declaration "*let* $n > 2$ *be an integer*". In this case, an identifier, "n", is introduced as an integer, and it is further restricted to be greater than 2 in the same formula it was introduced in. The semantic representation could be something like $\textbf{int}(n) \wedge \textbf{greater}(n, 2)$. The logical representation generated with our grammar actually looks a bit different (Section 3.3). Anyway, it becomes obvious that we need to identify and extract "n" in order to assert that it is an **int**. A different example is the sentence "*we conclude that* $n > 2$". In this case, the formula takes the place of a whole sub-sentence and there is no need to extract an identifier.

In these two scenarios, the formulae are represented by different GF categories: Identifier (with potential restrictions) and Statement. Figure 3 shows the parse tree of "$n > 2$" as an Identifier (as in the declaration "*let* $n > 2$ *be an integer*"). The root node combines the identifier "n", the relation ">", and the expression "2". Figure 4 shows the parse tree for "$n > 2$" as a Statement (as in "*we conclude that* $n > 2$"). Note that the trees differ depending on the type of formula.

Another noteworthy design decision was to represent the binary relation ">" simply as a leaf node. An alternative approach would have been to represent the relation as a function which takes two expressions as arguments. We decided against this, so that each relation can be used in different ways. This is particularly useful, because binary relations can be combined into relations of higher arity. For example, in "$0 < r \leq 1$", we could consider $\cdot < \cdot \leq \cdot$ as a ternary relation. Defining every possible combination of these higher-arity relations

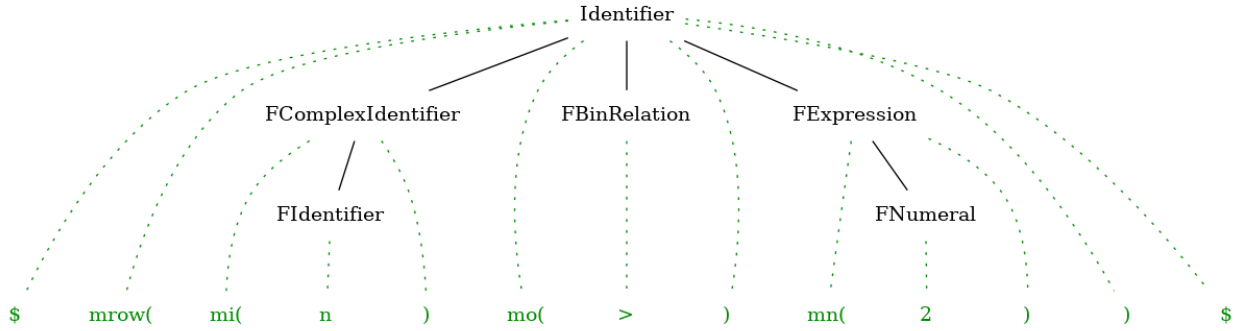


Figure 3: The parse tree for "$n > 2$" as a (restricted) Identifier. The nodes are labelled with the category names (i.e. types).

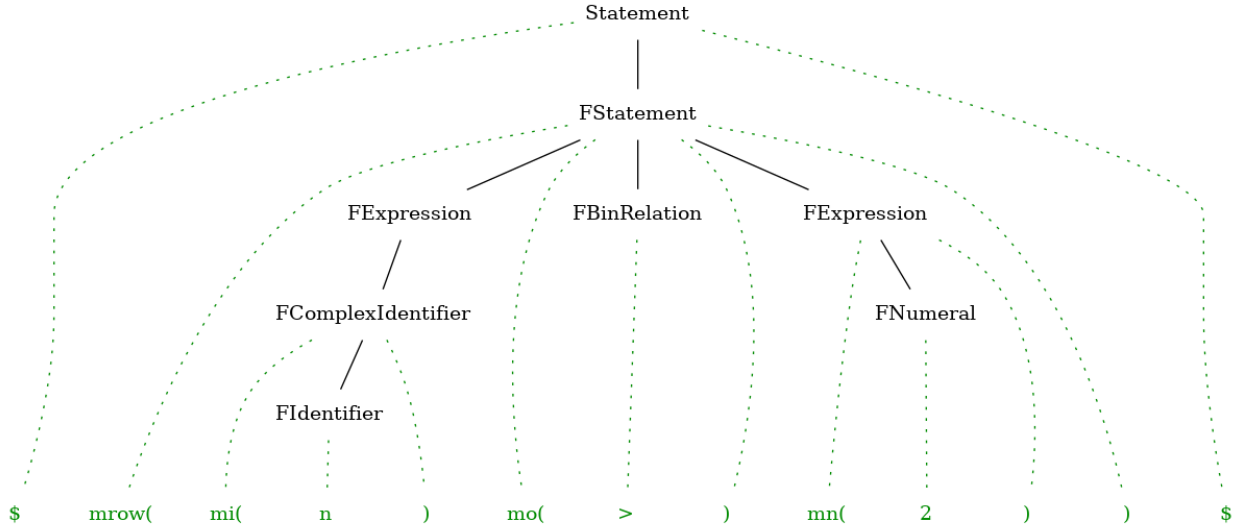


Figure 4: The parse tree for "$n > 2$" as a Statement. The nodes are labelled with the category names.

would be very difficult as they can get quite long – consider e.g. the relation "$0 = t_0 < t_1 < \ldots < t_n = 1$", in which things are complicated even further with an ellipsis.

3.3 The Discourse Grammar

The discourse grammar deals with the natural language in documents. It basically combines natural language rules (which we get from the resource grammar library) with mathematical vocabulary, and extends them with a set of idioms that are used in mathematical language.

3.3.1 Mathematical Objects

Let's first take a look at mathematical objects, the "building blocks" of mathematics. With this we refer to phrases like "*positive integer*" or "*bijection on the real numbers*". The descriptions of mathematical objects can get quite complex, as for example in the declaration "*let f be a continuous real-valued function defined on the interval [0,1] that is monotonically increasing on* $[\frac{1}{4}, \frac{3}{4}]$".

In GF, we represent mathematical objects with the category MObj. There are atomic MObjs like integer_MObj. More complex MObjs can be composed by putting further restrictions on the object. For example, in "*positive integer*" (parse tree in Figure 5), integer_MObj is further restricted with positive_MObjProp. The leaves integer_MObj and positive_MObjProp are defined in the mathematical lexicon.

With the concrete syntax for logical representations, "*positive integer*" gets linearized into the λ-function $\lambda x.\mathbf{pos}(x) \wedge \mathbf{int}(x)$. This way, the declaration "*let n be a positive integer*" can get linearized into $(\lambda x.\mathbf{pos}(x) \wedge$

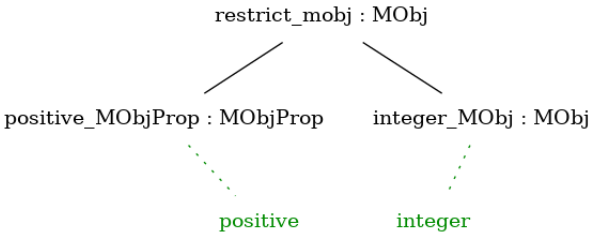


Figure 5: The parse tree for the MObj "*positive integer*".

$\mathbf{int}(x))n$, which β-reduces to $\mathbf{pos}(n) \wedge \mathbf{int}(n)$. Note that we interpreted the adjective "*positive*" intersectively, i.e. the set of positive integers is the intersection of the set of positive objects with the set of integers. Most mathematical adjectives are, with few exceptions: "*proper*" is not, but still subsective: the set of "*proper subgroup*"s is a subset of the set of "*subgroup*"s. Examples of mathematical adjectives that are not even subsective are "*improper*" and "*generalized*", but also the prefix "*quasi*" as in "*quasi-group*".

3.3.2 Definitions

One of the motivations behind this project is the formalization and translation of SMGloM [Gin+16], the "Semantic, Multilingual Glossary for Mathematics", which contains a large number of definitions for different mathematical fields in a variety of natural languages. A very simple example definition is "*a unital quasigroup is called a loop*". The parse tree for this definition is shown in Figure 6. It demonstrates many of the design decisions we've made so far. The root node combines two mathematical objects (MObjs, see also Section 3.3.1), which are the definiens ("*unital quasigroup*") and the definiendum ("*loop*"), as well as an Identifier (see Section 3.2), which can get assigned to the definiens. In this case, however, no identifier is provided, which is denoted by the special identifier no_identifier. For the logical representation, a default identifier, x_0, is introduced in this situation: $(\forall x_0.(\lambda x.\mathbf{unital}(x) \wedge \mathbf{quasigroup}(x))x_0 \Rightarrow ((\lambda x.\mathbf{loop}(x))x_0))$. Applying β-reduction to this expression, we get $\forall x_0.\mathbf{unital}(x_0) \wedge \mathbf{quasigroup}(x_0) \Rightarrow \mathbf{loop}(x_0)$.

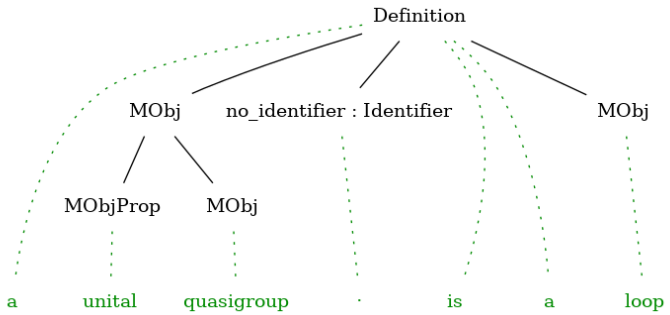


Figure 6: The parse tree for the definition "*a unital quasigroup is called a loop*". The nodes are labelled with the category names (types). no_identifer is a special Identifier indicating that no identifier was introduced.

Usually, definitions are more complex and have an additional condition. An example for this is the definition "*an integer n is called even, iff $2|n$*". The definition core "*an integer n is called even*" could grammatically be a complete definition on its own, but there is the additional condition "$2|n$". The parse tree (Figure 7) has the root node iff_definition, which takes a definition (core) and a Statement that is the additional condition. The generated logical representation for this definition is $(\forall n.((\lambda x.\mathbf{int}(x))n) \Rightarrow ((\lambda x.\mathbf{even}(x))n \Leftrightarrow \mathbf{divides}(2, n)))$ (or simplified via β-reduction $\forall n.\mathbf{int}(n) \Rightarrow (\mathbf{even}(n) \Leftrightarrow \mathbf{divides}(2, n))$).

4 Evaluation and Examples

The coverage of our framework is still limited to a few examples. Apart from the small examples shown above, a few larger examples are covered as well, like the previously mentioned definition of a prime number:

"*A positive integer n is called prime, iff there is no integer $1 < m < n$ such that $m|n$*".

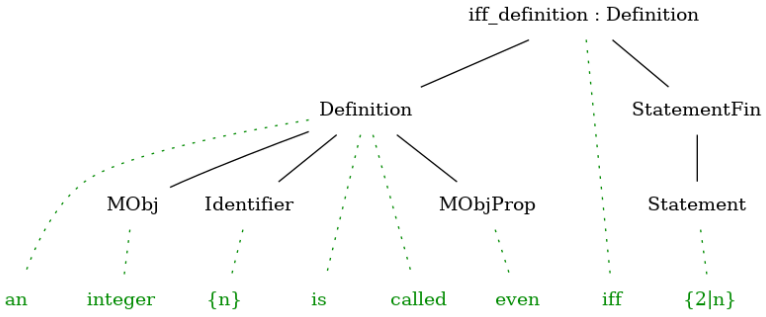


Figure 7: The parse tree for the definition "*an integer n is called even, iff* $2|n$". Subtrees of formulae were collapsed for conciseness.

Note that we used a shorthand notation of the formula representation for conciseness. The generated German translation is:

"*Eine positive ganze Zahl n ist prim genau dann, wenn es keine ganze Zahl* $1 < m < n$ *gibt, sodass* $m|n$".

Which is not the translation we suggested in the introduction, but definitely correct. The following logical representation is generated from the parse tree (which is too large to show here):

$(\forall n.((\lambda x.\mathbf{pos}(x) \wedge \mathbf{int}(x))n) \Rightarrow ((\lambda x.prime(x))n \Leftrightarrow (\neg\exists m.(\lambda x.\mathbf{int}(x))m \wedge \mathbf{less}(1,m) \wedge \mathbf{less}(m,n) \wedge (\mathbf{divides}(m,n)))))$

This can be β-reduced to the desired representation mentioned in the introduction:

$\forall n.\mathbf{pos}(n) \wedge \mathbf{int}(n) \Rightarrow (\mathbf{prime}(n) \Leftrightarrow \neg\exists m.\mathbf{int}(m) \wedge \mathbf{divides}(m,n) \wedge \mathbf{less}(1,m) \wedge \mathbf{less}(m,n))$

More examples and the source code can be found at [MGF].

5 Conclusion, Next Steps and Future Work

We have presented our ongoing efforts to use GF for the translation and formalization of mathematical documents. The syntactical framework is established and tested on a few example sentences, which were successfully translated between English and German and formalized into a logical representation.

It should be mentioned that, once English sentences could be parsed, it was surprisingly easy to create rudimentary German concrete syntaxes for translation (it took only about 2 hours to get a first translation of the prime number definition). Adding more languages could be even easier, if we better take advantage of GF's functors, which essentially allow to have shared code for the common parts of concrete syntaxes.

Some of the next steps would be:

- Switching to Discourse Representation Theory (DRT) [Kam81] for the logical representation. This would be required for the representation of anaphora, but could also simplify, for example, the handling of declarations.
- Extending the grammars for wider coverage. One of the more challenging extensions would be the handling of multiple identifiers in declarations (as e.g. in "*let* $n < m$ *be integers*"). Armin Fiedler and Xiaorong Huang's work on aggregation [FH95] might be useful for this.
- Using large, multilingual lexica for increased coverage.
- Using the framework (once it has sufficiently matured) for translations in SMGloM [Gin+16] (the Semantic, Multilingual Glossary for Mathematics).

The grammar, along with a few examples, can be found at [MGF].

References

[Arc+12] Dominique Archambault et al. "Using GF in multimodal assistants for mathematics". In: *Digitization and E-Inclusion in Mathematics and Science*. 2012, pp. 1–12. URL: https://www.molto-project.eu/sites/default/files/ArchambaultCaprottiRantaSaludes_0.pdf.

[Arxa] *arXMLiv*. seen July 2018. URL: https://kwarc.info/projects/arXMLiv/.

[Arxb] *arxiv.org e-Print archive*. URL: http://www.arxiv.org.

[Aus+03] Ron Ausbrooks et al. *Mathematical Markup Language (MathML) Version 2.0 (second edition)*. W3C Recommendation. World Wide Web Consortium (W3C), 2003. URL: http://www.w3.org/TR/MathML2.

[Bau99] Judith Baur. "Syntax und Semantik mathematischer Texte – ein Prototyp". MA thesis. Saarbrücken, Germany: Fachrichtung Computerlinguistik, Universität des Saarlandes, 1999.

[Bus+04] Stephen Buswell et al. *The Open Math Standard, Version 2.0*. Tech. rep. The OpenMath Society, 2004. URL: http://www.openmath.org/standard/om20.

[Cap] Olga Caprotti. *WebALT! Deliver Mathematics Everywhere*.

[CS12] Olga Caprotti and Jordi Saludes. "The Gf Mathematical Grammar Library". In: *24th OpenMath Workshop, 7th Workshop on Mathematical User Interfaces (MathUI), and Intelligent Computer Mathematics Work in Progress*. (Bremen, Germany, July 9–13, 2012). Ed. by James Davenport et al. CEUR Workshop Proceedings 921. Aachen, 2012, pp. 49–52. URL: http://ceur-ws.org/Vol-921/openmath-02.pdf.

[FH95] Armin Fiedler and Xiaorong Huang. "Aggregation in the Generation of Argumentative Texts". In: *Proceedings of the 5th European Workshop on Natural Language Generation*. Leiden, Netherlands: Rijks University Leiden, 1995, pp. 5–9.

[GF] *GF - Grammatical Framework*. URL: http://www.grammaticalframework.org (visited on 09/27/2017).

[Gin+16] Deyan Ginev et al. "The SMGloM Project and System. Towards a Terminology and Ontology for Mathematics". In: *Mathematical Software - ICMS 2016 - 5th International Congress*. Ed. by Gert-Martin Greuel et al. Vol. 9725. LNCS. Springer, 2016. DOI: 10.1007/978-3-319-42432-3. URL: http://kwarc.info/kohlhase/papers/icms16-smglom.pdf.

[Gin11] Deyan Ginev. "The Structure of Mathematical Expressions". Master's Thesis. Bremen, Germany: Jacobs University Bremen, Aug. 2011. URL: http://kwarc.info/people/dginev/publications/DeyanGinev_Mastersthesis.pdf.

[Hor99] Helmut Horacek. "Presenting Proofs in a Human-Oriented Way". In: *Proceedings of the 16th Conference on Automated Deduction*. Ed. by Harald Ganzinger. LNAI 1632. Springer Verlag, 1999, pp. 142–156.

[Hua89] Xiaorong Huang. "Proof Transformation Towards Human Reasoning Style". In: *Proceedings of the of 13th GWAI*. Ed. by Dieter Metzing. Informatik-Fachberichte 216. Springer Verlag, 1989, pp. 37–42.

[Kam81] Hans Kamp. "A Theory of Truth and Semantic Representation". In: *Formal Methods in the Study of Language*. Ed. by J. Groenendijk, Th. Janssen, and M. Stokhof. Amsterdam, Netherlands: Mathematisch Centrum Tracts, 1981, pp. 277–322.

[MGF] *The Mathematical Grammatical Framework*. URL: https://gl.kwarc.info/smglom/GF (visited on 07/14/2018).

[Mil] Bruce Miller. *LaTeXML: A LaTeX to XML Converter*. URL: http://dlmf.nist.gov/LaTeXML/.

[Ran04] Aarne Ranta. "Grammatical Framework — A Type-Theoretical Grammar Formalism". In: *Journal of Functional Programming* 14.2 (2004), pp. 145–189.

[Ran11] Aarne Ranta. *Grammatical Framework: Programming with Multilingual Grammars*. ISBN-10: 1-57586-626-9 (Paper), 1-57586-627-7 (Cloth). Stanford: CSLI Publications, 2011.

[Ran16] Aarne Ranta. *Grammatical Framework - Formalizing the Grammars of the World*. Sept. 7, 2016. URL: http://www.grammaticalframework.org/~aarne/gf-google-2016.pdf.

[SX11] Jordi Saludes and Sebastian Xambó. "The GF Mathematics Library". In: *THedu*. Ed. by Pedro Quaresma and Ralph-Johan Back. Vol. 79. EPTCS. 2011, pp. 102–110. DOI: 10.4204/EPTCS.79.6.

[VLP07] Konstantin Verchinine, Er Lyaletski, and Andrei Paskevich. "System for Automated Deduction (SAD): A Tool for Proof Verification". In: *Proceedings of the 21st International Conference on Automated Deduction, number 4603 in Lecture Notes in Artificial Intelligence*. SpringerVerlag, 2007, pp. 398–403.

[WKK04] Magdalena Wolska and Ivana Kruijff-Korbayová. "Analysis of Mixed Natural and Symbolic Input in Mathematical Dialogs". In: *ACL*. 2004, pp. 25–32.

[Wol13] Magdalena A. Wolska. "Student's Language in Computer-Assisted Tutoring of Mathematical Proofs". PhD thesis. ComputerLinguistik, Saarland University, 2013.

[Zin04] Claus Zinn. "Understanding Informal Mathematical Discourse". PhD thesis. Technischen Fakultät der Universität Erlangen-Nürnberg, 2004. URL: `https://sites.google.com/site/clauszinn/verifying-informal-proofs/37_04.pdf`.